On Using Markov Decision Processes to Model Integration Solutions for Disparate Resources in Software Ecosystems

Rafael Z. Frantz[1], Vitor Basto-Fernandes[2], Sandro Sawicki[1], Fabricia Roos-Frantz[1], Iryna Yevseyeva[3], Michael Emmerich[4]

[1]*UNIJUÍ University, Department of Exact Sciences and Engineering, Brazil*

[2]*Polytechnic Institute of Leiria, School of Technology and Management, Portugal*

[3]*Newcastle University, School of Computing Science, United Kingdom*

[4]*Leiden University, Leiden Institute of Advanced Computer Science, The Netherlands*

rzfrantz@unijui.edu.br, vitor.fernandes@ipleiria.pt, sawicki@unijui.edu.br, frfrantz@unijui.edu.br, iryna.yevseyeva@newcastle.ac.uk, emmerich@liacs.nl

Keywords: Simulation, Enterprise Application Integration, Domain-Specified Language, Markov Decision Process.

Abstract: The software ecosystem of an enterprise is usually composed of an heterogeneous set of applications, databases, documents, spreadsheets, and so on. Such resources are involved in the enterprise's daily activities by supporting its business processes. As a consequence of market change and the enterprise evolution, new business processes emerge and the current ones have to be evolved to tackle the new requirements. It is not a surprise that different resources may be required to collaborate in a business process. However, most of these resources were devised without taking into account their integration with the others, i.e., they represent isolated islands of data and functionality. Thus, the goal of an integration solution is to enable the collaboration of different resources without changing them or increasing their coupling. The analysis of integration solutions to predict their behaviour and find possible performance bottlenecks is an important activity that contributes to increase the quality of the delivered solutions. Software engineers usually follow an approach that requires the construction of the integration solution, the execution of the actual integration solution, and the collection of data from this execution in order to analyse and predict their behaviour. This is a costly, risky, and time-consuming approach. In this paper we discuss the usage of Markov models for formal modelling of integration solutions aiming at enabling the simulation of the conceptual models of integration solution still in the design phase. By using well-established simulation techniques and tools at an early development stage, this new approach contributes to reduce cost, risk, development time and improve software quality attributes such as robustness, scalability and maintenance.

1 INTRODUCTION

Enterprises rely on integration solutions to support their business processes by promoting the reuse of resources available in their software ecosystem (Messerschmitt and Szyperski, 2003), which is usually composed of an heterogeneous set of applications, databases, documents, spreadsheets, and so on. The success of such business processes is highly dependent on the correct and efficient execution of the integration solutions. All over the years, several technologies have emerged to support the development of integration solutions, chiefly by providing methodologies and tools to design, implement, and run integration solutions. The state-of-the-art technologies have been pushing the development of integration solutions from a code centric to a model centric approach, by providing domain-specific languages that enable the design of conceptual models at a high level of abstraction. In this paper we are interested in the analysis of the conceptual models designed for integration solutions using the domain-specific language of Guaraná technology. The goal of an integration solution is to enable the collaboration of different resources that were devised without taking into account their integration with the others, i.e., they represent isolated islands of data and functionality. Thus, an integration solution enables data synchrony and the development of new functionalities on top of the actual resources (Hohpe and Woolf, 2003).

The analysis of integration solutions to predict their behaviour and find possible performance bot-

tlenecks is an important activity that contributes to increase the quality of the delivered solutions. It is not enough to design a conceptual model for an integration solution, but it is also essential to analyse its behaviour under different workloads and minimise performance bottlenecks. Most often, the approach adopted by software engineers requires the construction of the integration solution, the execution of the actual integration solution, and the collection of data from this execution. This is a costly, risky, and time-consuming approach. The construction of integration solutions is expensive because it demands a good command on an integration technology to map the conceptual model into executable code. The execution involves the setup of a controlled environment in which the integration solution can be deployed, the generation and injection of input data into the integration solution, and the emulation of critical running scenarios. Any faults in the constructed solution may cause the execution to fail and negatively affect the analysis. The collection of data requires the insertion of extra code into the constructed integration solution and the configuration of the runtime system of the adopted integration technology that enables monitoring and data collection. A new approach that enables the analysis of the behaviour and discovering possible performance bottlenecks still in the design phase, taking as input the conceptual models of integration solutions, would help to reduce cost, risk, and development time.

In this context, enterprise integration solutions shall be understood as discrete event systems. Discrete models are event-oriented and so used to model systems that change their state at discrete moments in time according to the occurrence of events. Integration solutions can be characterised as discrete systems for the reason that all components involved in an integration solution consume a particular execution time when an event occurs. Thus, any event can change the state of the integration solution.

As a discrete system, the conceptual model designed for an integration solution can be simulated taking the advantage of well-established techniques and tools for discrete-event simulation. It would allow software engineers to focus on performance measures that allow for discovering possible problems before deploying the integration solution into production and the commitment of capital and resources. The simulation of integration solutions would help to prevent unexpected behaviours and cut down cost, risk, and time to deliver better integration solutions.

High-level state-based modelling languages available in the integration technologies provide description syntaxes for model construction, ability to compute the reachable state space, indication if specific properties are satisfied by the model and other quantitative results relevant to identify interesting patterns or trends in the behaviour of a system. Model checking, simulation and experiments configuration support allows for the analysis of system properties as functions of model and property parameters. In this paper we discuss the use of Markov models for formal modelling of integration solutions, chiefly Markov Decision Process models, which allow for both probabilistic and nondeterministic modelling.

The rest of this paper is organised as follows: Section 2 discusses the related work that has also used discrete-event simulation to analyse the behaviour and discover possible performance bottlenecks in actual systems; Section 3, provides an overview of Guaraná domain-specific language that we use to design the integration solution conceptual models; Section 4, discusses the use of Markov models to enable simulation of discrete systems; Section 5 presents a case study in which we introduce a conceptual model designed using Guaraná and its corresponding formal model using Markov Decision Process; and, finally, Section 6 presents our main conclusions.

2 RELATED WORK

There are some previous work in the literature that have used discrete-event simulation techniques and tools to analyse systems to predict their behaviour and find possible performance bottlenecks. Nevertheless, from the literature there is no evidence that discrete-event simulation has been explored aiming at the analysis of conceptual models of enterprise integration solutions. There is a work that has used discrete-event simulation in the field of EAI, however in this work Janssen and Cresswell (2005) focus on organisational issues and the interests of stakeholders, by providing a simulation-based methodology to evaluate the impact of enterprise integration at business level before its implementation. They argue that the commitment of stakeholders is one of the keys for the success of EAI, and thus make stakeholders central to their methodology. Their research analyses integration problems only in the context of public organisations, in which they use discrete-event simulation to quantify the benefits that the integration could bring to the organisation at business level.

The proposal presented in this paper differs from theirs since its focus is on the infrastructure technology used to design and implement integration solutions. Another work, presented by Al-Aomar (2010), describes the basic structure of service system simu-

lation using application case studies targeting the performance of the service system. The author identifies and defines the main characteristics and elements of service system as system entities, service providers and customer service. In addition, he explores the modelling techniques that are used in the development of discrete-event simulation models targeting service systems. This work considers model elements, model logic, model data, model parameters, decision variables, and performance measures in its case studies simulation. The number of entities that arrive in the specified time interval is a random variable that follows Poisson distribution. Arrival rates and service rates are essential to calculate system performance measures. In this study the author collects the arrival and service rates, which are attributes used to represent customers arriving at a service centre. For example, for each customer, the times of arrival, service start, and service end are calculated. Having these times collected, the Time Between Arrival and Service Time can also be computed. Time Between Arrival is the time since last arrival and Service Time is the total time used by a service. After, Mean Time Between Arrivals and Mean Service Time are used to compute the average time. By using this strategy to collect inter-arrival and service times, it is possible to find bottlenecks located at services and queues.

The article presented by Desa et al. (2013), demonstrates the potential of discrete-event simulation for detecting bottlenecks. The authors developed a discrete-event simulation model based on the logic and using data collected from a manufacturing plant specialised in producing aircraft parts. In this work, the detection of bottlenecks was based on resource utilisation and work in process. Arena simulation software was employed to model and analyse the several activities. With the model proposed in this study, it was possible to understand and improve the performance of the production system; furthermore, it was found that the discrete-event simulation is capable of analysing the behaviour of complex and sophisticated systems. Kunz et al. (2011) present a performance prediction methodology that calculates the best possible performance bound for expanded parallel discrete-event simulations. According to the authors, predicting and analysing runtime performance features and understanding its behaviour is an important step in the development process of parallel discrete-event simulations. Their methodology is based on linear programming that calculates an optimal event execution schedule for a given simulation and a set of microprocessors. They use linear programming for predicting the runtime performance of a parallel simulation model. A trace is used to the linear program that computes an optimal event schedule targeting to minimise the overall simulation runtime. Discrete-event simulation model can also be used to an automated bottleneck analysis to detect running production constraints. Faget et al. (2005) present a case study that was conducted in Toyota Company. The authors describe the application of a method for detecting bottlenecks using discrete-event models and design of experiments to suggest improvement alternatives. They also report that the main challenge was to educate the decision makers about the importance of simulation as a support tool to analyse the behaviour of production systems. The results obtained with the simulation had better accuracy in bottleneck analysis and provide useful information for improvements.

3 GUARANÁ TECHNOLOGY

Guaraná technology supports software engineers in the design, implementation, and execution of integration solutions. Since the focus of this paper is on the simulation of conceptual models, this section aims to provide a brief overview of the modelling language provided by this technology. In Guaraná, the conceptual models are designed using a domain-specific language, which has an easy-to-learn and intuitive graphical notation. This language is based on the integration patterns documented by Hohpe and Woolf (2003) and enables the design of platform-independent models, i.e., the resulting models are not committed to a particular implementation technology (Frantz et al., 2010). The following integration concepts are supported by constructors in Guaraná language:

Message: An abstraction of a piece of information that is exchanged and transformed across an integration solution. It is composed of a header, a body, and one or more attachments. The header includes custom properties and frequently the following pre-defined properties: message identifier, correlation identifier, sequence size, sequence number, return address, expiration date, and message priority. The body holds the payload data, whose type is defined by the template parameter in the message class. Attachments allow messages to carry extra pieces of data associated with the payload, e.g., an image or an e-mail message.

Task: Represents an atomic operation that can be executed on messages, such as split, aggregate, translate, chop, filter, correlate, merge, resequence, replicate, dispatch, enrich, slim, promote, demote, and delay. Roughly speaking, a task may have one or more inputs from which it receives

messages, and one or more outputs by means of which messages depart. Depending on the kind of operation, a task may be stateless or stateful. In a stateless task, the completion of its operation does not depend on previous or future messages; contrarily, the operation of a stateful task depends on previous or future messages to be completed, such as the case of the aggregator task, which has to collect the different correlated inbound messages to produce a single outbound message. In this paper, stateful tasks are not considered because the vast majority of tasks used in integration solutions are stateless.

Slot: A buffer connecting an input of one task to the output of another task aiming at messages to be processed asynchronously by tasks. A slot can follow different policies to serve messages to tasks, such as a priority-based output or a first-come, first-served. If a priority is defined in the message, slots follow the former policy; otherwise, the latter policy is adopted. In this paper, it is assumed that messages have no priority and the slot serves them in a first-come, first-served policy.

Port: Abstracts away from the details required to interact with an resources within the software ecosystem. Roughly speaking, by means of a port it is possible to establish read, write, solicit, and respond communication operations with the resources being integrated.

Integration Process: Contains integration logic that executes transformation, routing, modification, and time-related operations over messages. An integration process is composed of ports that allow it to communicate with the resources being integrated, slots and a set of tasks to specify the integration logic.

Conceptually, an integration solution aggregates one or more integration processes through which messages flow and are processed asynchronously. The integration flow is actually implemented as a Pipe and Filter architecture, in which the pipes are implemented by Slots and the filters are implemented by Tasks. Every task realises an integration pattern (Hohpe and Woolf, 2003) and its execution depends on the availability of messages in all slots connected to its inputs. Slots are key constructors to enable asynchrony in an integration solution, thus messages are stored on them until they can be read by the next task in the integration flow. A detailed discussion on the domain-specific language provided by Guaraná is presented in Frantz et al. (2011).

Table 1 shows the concrete syntax for the constructors in Guaraná. Since the domain-specific language of Guaraná provides support to several integration patterns, the symbol that we depict in Table 1 to represent them is generic. Note the small rounded connectors on the sides of the icon; they represent the inputs and the outputs. Slots are connected to tasks using these rounded connectors. Note that, in this table we have included the notation we use to represent the resources (such as applications, databases, documents, and spreadsheets) being integrated. In the table, this notation does not specify which layer (database, channel, file, API, user interface, and so on) is being used to communicate the integration solution to the resource. However, software engineers can replace the doted circle containing an "X" by well-known icons, cf. Figure 1. Messages do not appear in this table, because they are not part of the conceptual model, they only exist and flow in the constructed and running integration solution.

4 SIMULATION MODEL

In this paper, a formal modelling based on probabilistic extensions of temporal logics is also addressed, aiming at the characterisation of integration solution as solutions that exhibit stochastic behaviour, operate under constraints on timing and other resources. Model checking tools based on probabilistic extensions of temporal logics are introduced for formal and quantitative model verification, bottleneck detection and performance analysis.

A stochastic process represents in this context the evolution of a system of random variables over time. In other words, a process with some degree of uncertainty, that evolves through a set of possible directions, starting from one of the possible states.

Stochastic model simulation in general and model checking in particular is an important component in the design and analysis of critical software systems, with probabilistic, nondeterministic and real time characteristics. Model checkers are able to analyse if a system satisfies a state/transition model and a temporal logic specification. In addition to conventional system analysis, stochastic model checking calculates the likelihood of certain events during system execution, including temporal relationship considerations (Kwiatkowska et al., 2011).

Amongst several existing probabilistic models, discrete-time Markov chains, which allow to specify the probability of making a transition from one state to another, continuous-time Markov chains, which allow to model continuous real time and probabilistic

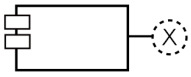
Notation	Concept	Notation	Concept
	Resource		Solicitor Port
	Integration Process		Responder Port
	Entry Port		Task
	Exit Port		Slot

Table 1: Guaraná concrete syntax.

choice, and Markov Decision Processes, which allow for both probabilistic and nondeterministic modelling, are of special interest for integration solution formal modelling. In Markov models, future states are influenced only by the current state and no influence exist from past states. Markov based simulation and model checking allows for systems properties analysis, such as path-based, transient and steady-state properties (Parker, 2011).

Discrete-time Markov Chains can be described as state-transition models augmented with probabilities. More formally, as a tuple of a finite set of states (S), an initial state s0 belonging to S, a transition probability matrix (P) of $S \times S \rightarrow [0-1]$ where transitions sum from a state must be 100%, a labelling function assigning to states a set of atomic propositions ($L : S \rightarrow 2^{AP}$). Time is commonly seen in Discrete-time Markov Chain models as discrete time-steps, homogeneous, and transition probabilities are independent of time. An execution of a Discrete-time Markov Chain is represented by a path, with length equal to path transitions. Paths probabilities calculation allow for system behaviour analysis and reason on properties such as probabilistic reachability. Quantitative and qualitative properties, including repeated reachability and persistence, might be of interest in the context of a system behaviour analysis, and addressed with Discrete-time Markov Chain probabilistic model checking. Reward structures modelling can also be used to represent benefit or cost characteristics (e.g. energy consumption). Transition rewards (instantaneous) and state rewards (cumulative) are modelled by the reward functions $rs : S \rightarrow R_{\geq 0}$ and $rt : S \times S \rightarrow R_{\geq 0}$ respectively. A reward structure can be used to measure the number of time-steps spent in a state or the chance that the system is in a specific state after a certain number of time-steps (Kwiatkowska et al., 2011; Parker, 2011).

Continuous-time Markov Chains main differences from Discrete-time Markov Chains models rely on the fact that transitions occur in real time and transitions are represented in a transition rate matrix ($R : S \times S \rightarrow R_{\geq 0}$), assigning rates to pairs of states (with probability of transition being triggered within t time-units equal to $1 - e^{-R(s,s)\cdot t}$), instead of giving the probability of making a transition between states. If in a state s, there is more than one state s' for which $R(s,s') > 0$ (known as a race condition), the first transition to be triggered determines the next state of the Continuous-time Markov Chain. Paths are sequences of states with time attributes (states execution duration). Transient and steady-state behaviour are properties of high interest and analysis in Continuous-time Markov Chains models, they address the likelihood of a system to be in a state at a specific time (or in the long-run), and time spent in specific system states. Reward structures are used to analyse number of requests served in a time interval or in the long-run, as well as the analysis of queue sizes at any time instant or in the long-run (Kwiatkowska et al., 2011; Parker, 2011).

Markov decision processes, like Discrete-time Markov chains, model a system as a discrete set of states and transitions between states occurring in discrete time-steps. In addition, Markov decision process models extend Discrete-time Markov chains by allowing for nondeterministic choice. This type of modelling is specially suited for concurrency, un-

known environments, and underspecification application scenarios. Since Discrete-time Markov chain models are fully probabilistic, they are unable to address some aspects of a system, such as nondeterministic choice(Parker, 2011).

Formally, a Markov decision process is a tuple (S, s_{init}, Steps, L) where S is a finite set of states, $s_{init} \in S$ is the initial state, Steps : $S \to 2^{Act \times Dist(S)}$ is the transition probability function, Act a set of actions, $Dist(S)$ the set of discrete probability distributions over S, and $L : S \to 2^{AP}$ a labelling with atomic propositions. A path in a Markov decision process is a sequence of states and action/distribution pairs, e.g., $s_0(a_0,\mu_0)s_1(a_1,\mu_1)s_2$, representing a system execution. Paths resolve nondeterministic (turn the model into a Discrete-time Markov Chain) and probabilistic choices, and then calculate a probability measure over paths(Kwiatkowska et al., 2011; Parker, 2011).

In Markov decision processes, adversaries (also known as strategies or policies) are used to resolve nondeterministic choices (multiple distributions) and may belong to different classes such as memoryless, finite-memory, randomised, fair. An adversary is a function mapping every finite path to an element of the Markov decision process model Steps. Markov decision processes provide best/worst case analysis based on lower/upper bounds on probabilities over all possible adversaries (Kwiatkowska et al., 2011; Parker, 2011).

For integration solutions formal model checking and performance analysis, Markov decision processes seem to be the most suitable type of probabilistic models to adopt. This type of probabilistic modelling has extensive support on modern probabilistic model checker software tools such as PRISM (Oxford, 2014). PRISM is a free and open source probabilistic model checker software, for formal modelling and analysis of systems in many different application domains (e.g. distributed systems). It is widely used in software, communications, distributed and embedded systems research, having a strong support from the formal methods research community.

5 CASE STUDY

Processing orders is a very common task. This section presents a well-known integration problem introduced by Hohpe (2005) to check the availability of order items. In this context, there are cases in which items listed in multi-item orders have to be processed separately because they are served by different inventories.

The software ecosystem in this case study involves five applications, namely: Ordering System, Widget Inventory, Gadget Inventory, Invalid Items Log, and Inventory System. Apart from the Widget Inventory and the Gadget Inventory, which provide an application programming interface that enables the communication with them, the interaction with the remaining applications has to be done by means of their database. Orders are placed by customers at the Ordering System. Because an order may be composed of several items, every order has to be split to check the availability of its items in the Widget Inventory or the Gadget Inventory. Items that do not belong to any of these inventories cannot be checked, so that they have to be logged in the Invalid Items Log. After checking the availability of every item in an order, they have to be aggregated into a single order to be processed by the Inventory System.

The following sections present a conceptual and a formal model. The former is designed using Guaraná Technology and the latter using Markov decision process notation.

5.1 Conceptual model

This section introduces a conceptual model designed to represent a possible integration solution for this case study. Figure 1, presents this model using the domain-specific language of Guaraná technology.

The workflow begins at entry port `P1`, which periodically polls the `Ordering System` to find new orders, and ends by writing messages with information regarding their availability to the `Inventory System`. Inbound messages polled by entry port `P1` are written to slot `S1`, which is the input for task `T1`.

Every message with a new order is split by task `T1` into individual messages, each of which contain only one item; the resulting messages are written to slot `S2`, which is the input of task `T2` and desynchronises tasks `T1` and `T2`. Messages in slot `S2` are analysed by task `T2` and then dispatched either in direction of the `Widget Inventory` or the `Gadget Inventory`. Messages with items that do not belong to any of these inventories, are routed to the `Invalid Items Log` and written to this system by means of port `P2`.

Task `T3` replicates the message aimed at `Widget Inventory`, so that one copy can be translated into a query message by task `T4` and the other copy (base message) is used by task `T5` to find the correlated response. The interaction with `Widget Inventory` to check the availability of items is made by means of port `P3`, which reads query messages from slot `S3` and writes the responses to slot `S4`. Correlated messages are read by task `T6` from its inbound slots and the information about the item availability is then

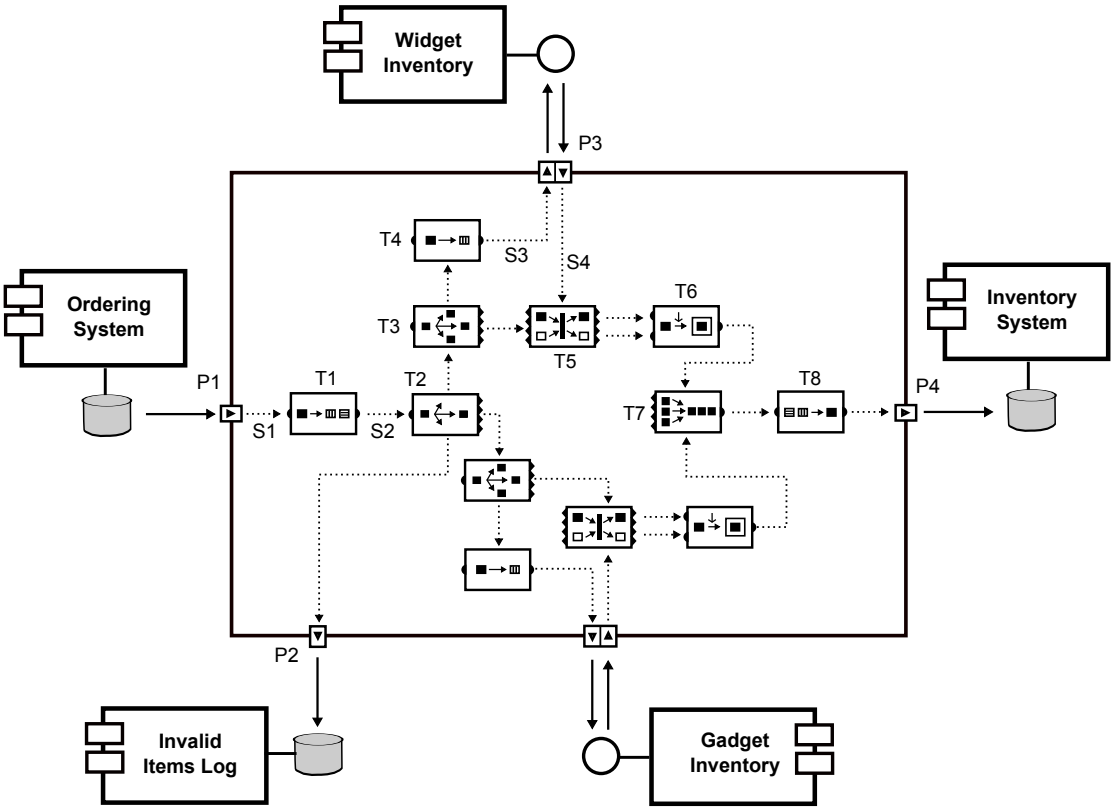


Figure 1: A conceptual model using Guaraná.

used to enrich the content of the base message, which keeps going in the flow. The interaction with `Gadget Inventory` occurs symmetrically.

Base messages now having the responses of availability for every item in the original order are merged by task `T7` into a single slot and then aggregated again into a single order message by task `T8`, and written to the `Inventory System` by means of port `P4`.

5.2 Markov decision process model

A simplified and partial Markov decision process model of the integration solution presented in Figure 1 is shown in Figure 2, to illustrate schematically the modelling ability of Markov decision processes in the EAI domain.

This model can be specified in PRISM modelling language for model checking and analysis. Markov decision process concepts are directly supported by PRISM modelling concepts and are introduced next. The concept of module in PRISM represents system processes, including process variables, which describe system states and commands (process behaviour, i.e., the way states change over time) comprising a guard (condition referring to variables of this or other module, required for the update to take place) and one or more updates together with the corresponding probabilities (updates can only affect variables belonging to the module). A PRISM model is constructed as the parallel composition of its modules. Variables can be of Boolean, Integer or Clock type. PRISM provides also full support for concepts such as labels, atomic prepositions, paths, rewards (transition and state rewards), invariants, race conditions, steady state and transient behaviour, defined in Markov decision processes models (Oxford, 2014).

Since it is not the purpose of this document to present the full PRISM syntax and functionality, only the simplified formal specification of Figure 2 Markov decision process model is described, as the basis for the PRISM model construction. Assuming a Markov decision process as a tuple M = (S, sinit, Steps, L), the example presented in Figure 2 can be described as:

$S = s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}$, the finite set of states, or state space.

$s_{init} = s_0$, the initial state.

$AP = \{order\_received, items\_classified,$ *items_routed, items_requested, items_rejected, items_ready, service_characterised, or-*

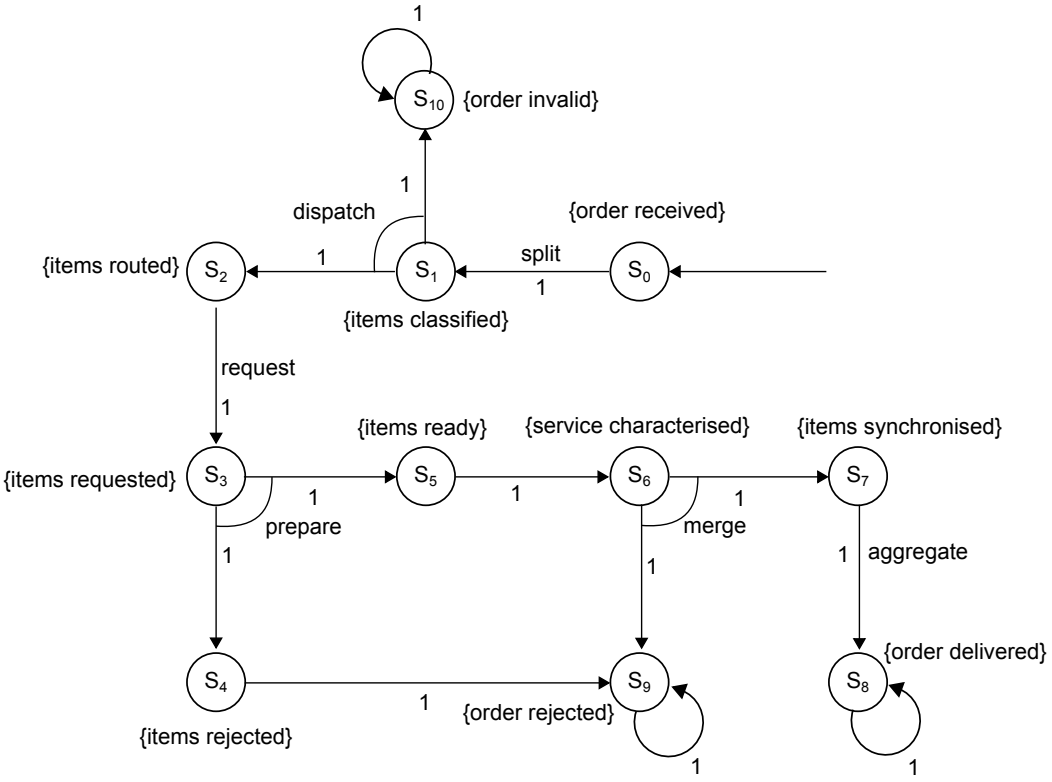


Figure 2: Markov decision process model of a simple (partial) integration solution.

der_rejected, order_served, order_invalid}, the set of atomic propositions.

L , the labelling with atomic propositions ($L : S \rightarrow 2^{AP}$), which are simply associated to the states in the current example.

$Steps : S \rightarrow 2^{Act \times Dist(S)}$, the transition probability function where Act is a set of actions {split, dispatch, replicate, transform, correlate, enrich, merge, aggregate} and $Dist(S)$ is the set of discrete probability distributions over the set S. In S all actions were simply associated probability 1, with action "merge" and "prepare" being represented as nondeterministic choices.

6 CONCLUSION

There is a high cost, risk, and development time spent associated with the simulation approach frequently adopted by software engineers to analyse the behaviour and find possible performance bottlenecks in enterprise integration solutions. This occurs due to the activities related with the construction, execution, and the collection of data from the execution of integration solutions. This paper proposes a new approach, based on discrete-event simulation, to cutting down cost, risk, and time to deliver better integration solutions. In this paper, modelling integration solutions as discrete-event systems was proposed in order to enhance the ability of software engineers to analyse not only the functional correctness of an integration solution (e.g. deadlocks detection), but also its non functional properties such as performance and resources usage. This extended formal analysis is proposed to be done by the means of Markov decision process models with the support of state of the art simulation tools, such as PRISM probabilistic model checker. A simple and representative integration solution was modelled with Guaraná domain specific language and its Markov decision process based formal model derived and described. This proposal addresses a major concern in the integration solutions software development life cycle and presents a scientifically innovative approach in the enterprise integration domain.

ACKNOWLEDGEMENT

The research work on which we report in this paper is supported by CAPES, FAPERGS, and the internal Research Programme at UNIJUÍ University.

REFERENCES

Al-Aomar, R. (2010). Simulating service systems. In Got, A., editor, *Discrete Event Simulations*, pages 1–25. In-Tech.

Desa, W. L. H. M., Kamaruddin, S., Nawawi, M. K. M., and Khalid, R. (2013). Evaluating the performance of a multipart production system using discrete event simulation (DES). In *International Proceedings of Economics Development & Research*, pages 64–67.

Faget, P., Eriksson, U., and Herrmann, F. (2005). Applying discrete event simulation and an automated bottleneck analysis as an aid to detect running production constraints. In *Proceedings of the 37th Conference on Winter Simulation*, pages 1401–1407.

Frantz, R. Z., Molina-Jimenez, C., and Corchuelo, R. (2010). On the design of a domain specific language for enterprise application integration solutions. In *International Workshop on Model-Driven Engeneering*, pages 19–30.

Frantz, R. Z., Reina-Quintero, A. M., and Corchuelo, R. (2011). A Domain-Specific language to design enterprise application integration solutions. *International Journal of Cooperative Information Systems*, 20(2):143–176.

Hohpe, G. (2005). Your coffee shop doesn't use two-phase commit. *IEEE Software*, 22(2):64–66.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.

Janssen, M. and Cresswell, A. M. (2005). An enterprise application integration methodology for e-government. *Journal of Enterprise Information Management*, 18(5):531–547.

Kunz, G., Tenbusch, S., Gross, J., and Wehrle, K. (2011). Predicting runtime performance bounds of expanded parallel discrete event simulations. In *IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 359–368.

Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0: Verification of Probabilistic Real-Time Systems. In Gopalakrishnan, G. and Qadeer, S., editors, *Computer Aided Verification*, pages 585–591. Springer Berlin Heidelberg.

Messerschmitt, D. and Szyperski, C. (2003). *Software EcoSystemm: Understanding an Indispensable Technology and Industry*. MIT Press.

Oxford (2014). Oxford University - PRISM Manual version 4.2.

Parker, D. (2011). Lectures - Probabilistic Model Checking, Department of Computer Science, University of Oxford.